Graphical Models as Block-Tree Graphs

Divyanshu Vats and José M. F. Moura

Abstract

We introduce *block-tree graphs* as a framework for deriving efficient algorithms on graphical models. We define block-tree graphs as a tree-structured graph where each node is a cluster of nodes such that the clusters in the graph are disjoint. This differs from junction-trees, where two clusters connected by an edge always have at least one common node. When compared to junction-trees, we show that constructing block-tree graphs is faster and finding optimal block-tree graphs has a much smaller search space. For graphical models with boundary conditions, the block-tree graph framework transforms the boundary valued problem into an initial value problem. For Gaussian graphical models, the block-tree graph framework leads to a linear state-space representation. Since exact inference in graphical models can be computationally intractable, we propose to use spanning block-trees to derive approximate inference algorithms. Experimental results show the improved performance in using spanning block-trees versus using spanning trees for approximate estimation over Gaussian graphical models.

Index Terms

Graphical Models, Markov Random Fields, Belief Propagation, Recursive Representation, Junction-Tree, Kalman Filter, Graphical Approximation, Block-Tree Graphs, Gaussian Graphical Models, Smoothing, Optimal Estimation

The authors are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213, USA (email: dvats@andrew.cmu.edu, moura@ece.cmu.edu, ph: (412)-268-6341, fax: (412)-268-3980).

I. INTRODUCTION

A graphical model is a random vector defined on a graph such that each node represents a random variable (or multiple random variables), and edges in the graph represent conditional independencies. The underlying graph structure in a graphical model leads to a factorization of the joint probability distribution. This property has lead to graphical models being used in many applications such as sensor networks, image processing, computer vision, bioinformatics, speech processing, and ecology [1], [2], to name a few. This paper derives efficient algorithms on graphical models. The structure of the graph plays an important role in determining the complexity of these algorithms. Tree-structured graphs are suitable for deriving efficient inference and estimation algorithms [3]. *Inference* in graphical models corresponds to finding marginal distributions given a joint probability distribution. *Estimation* of graphical models corresponds to performing inference over the conditional distribution $p(\mathbf{x}|\mathbf{y})$, where $\mathbf{x}$ is a random vector defined on a graph with noisy observations $\mathbf{y}$. State-space models can be interpreted as graphical models defined on a chain or a tree [4], [5], for which efficient estimation algorithms include the Kalman filter [6] or recursive smoothers [7]. Estimation and inference in arbitrary chain or tree structured graphical models is achieved via belief propagation [3]. These graphical models, however, have limited modeling capability [8], and it is thus desirable to consider more general graphs, i.e., graphs with cycles, an example of which is shown in Fig. 1(a).

A popular method for inference in graphs with cycles is to perform variable elimination, where the joint probability distribution is marginalized according to a chosen elimination order, which is a permutation of the nodes in the graph. Frameworks for variable elimination have been proposed in [9]–[11]. A general framework for variable elimination is achieved by constructing a junction-tree [12], which is a tree-structured graph with edges between clusters of nodes. The key properties of junction-trees are highlighted as follows:

(i) **Clusters in a junction-tree:** Two clusters connected by an edge in a junction-tree always have at least one common node. The number of nodes in the cluster with maximum size minus one is called the width of a graph, denoted as $\mathrm{w}(G)$ for a graph G.

(ii) **Constructing junction-trees:** This consists of two steps: triangulation, which has complexity $O(n)$, and a maximum spanning tree algorithm, which has complexity $O(m^2)$, where m is the number of cliques (see Section II-A) in a triangulated[1] graph [13]. The number of cliques m depends on the connectivity of the graph: if a graph is dense (many edges), m can be small and if a graph is

---

[1] A graph is triangulated if all cycles of length four or more have an edge connecting non-adjacent nodes in the cycle

sparse (small number of edges), m can be as large as $n - 1$.

(iii) **Optimal junction-trees:** For a graph G, there can be many different associated junction-trees. An *optimal junction-tree* is the one with minimal width, called the treewidth of the graph [14], denoted as $\text{tw}(G)$. Finding the optimal junction-tree, and thus the treewidth of a graph, requires a search over at most $n!$ number of possible combinations, where n is the number of nodes in a graph.

(iv) **Complexity of inference:** Inference in graphical models using junction-trees can be done using algorithms proposed in [12], [15], [16]. The complexity of inference using junction-trees is exponential in the treewidth of the graph [16].

From the above analysis, it is clear that constructing junction-trees can be computationally difficult. Further, finding optimal junction-trees is hard because of the large search space. Since finding optimal junction-trees is hard, finding the treewidth of a graph is also hard [17]. Thus, the complexity of inference using junction-trees really depends on the upper bound on treewidth computed using heuristic algorithms, such as those given in [18].

In this paper, we introduce *block-tree graphs*, as an alternative framework for constructing tree-structured graphs from graphs with cycles. The key difference between block-trees and junction-trees is that the clusters in a block-tree graph are disjoint, whereas clusters in a junction-tree have common nodes. We use the term block-tree because the adjacency matrix for block-tree graphs is block-structured under a suitable permutation of the nodes. The key properties of block-tree graphs and its comparison to the junction-tree are outlined as follows:

(i') **Clusters in a block-tree:** Clusters in a block-tree graph are always disjoint. We call the number of nodes in the cluster with maximum size the block-width, denoted as $\text{bw}(G)$ for a graph G.

(ii') **Constructing block-trees:** We show that a graph can be transformed into a block-tree graph by appropriately clustering nodes of the original graph. The algorithm we propose for constructing block-tree graphs involves choosing a root cluster and finding successive neighbors. An important property is that a block-tree graph is uniquely specified by the choice of the root cluster. Thus, constructing block-tree graphs only requires knowledge of a root cluster, which is a small fraction of the total number of nodes in the graph. On the other hand, constructing junction-trees requires knowledge of an elimination order, which is a permutation of all the nodes in the graph. Constructing a block-tree graph has complexity $O(n)$, where n is the number of nodes in the graphs. When compared to junction-trees, we avoid the $O(m^2)$ computational step, which is significant savings when m is as large as n.

(iii') **Optimal block-trees:** Different choices of root clusters result in different block-tree graphs. We define an *optimal block-tree graph* as the block-tree graph with minimal block-width, which we call the *block-treewidth* of a graph, denoted as btw(G). We show that computing the optimal block-tree, and thus the block-treewidth, requires a search over $\binom{n}{\lceil n/2 \rceil}$ possible number of choices. Although possibly very large for large n, this number is much less than $n!$, the search space for computing optimal junction-trees.

(iv') **Complexity of inference:** We show that the complexity of using block-tree graphs for inference over graphical models is exponential in the maximum sum of cluster sizes of adjacent clusters.

From (i') $-$ (iii'), we see that constructing block-tree graphs is faster and finding optimal block-tree graphs has a smaller search space. In general, the complexity of inference using block-tree graphs is higher, however, we show that there do exist graphical models for which the complexity of inference is the same for both the junction-tree and the block-tree graph.

Using disjoint clusters to derive efficient algorithms on graphical models has been considered in the past, but only in the context of specific graphical models. For example, [19] and [20] derive recursive estimators for graphical models defined on a 2-D lattice by scanning the lattice horizontally (or vertically). For specific directed graphs, the authors in [21] and [22], used specific disjoint clusters for inference. To our knowledge, previous work has not addressed questions like optimality of different structures or proposed algorithms for constructing tree-structured graphs using disjoint clusters. Our block-tree graphs address these questions for any given graph, even non-lattice graphs and arbitrary directed graphs.

Applying our block-tree graph framework to undirected graphical models with boundary conditions, such that the boundary nodes connect to the undirected components in a directed manner, we convert a boundary valued problem into an initial value problem. Motivation for using such graphs, which are referred to as chain graphs in the literature [23]–[25], is in accurately modeling physical phenomena whose underlying dynamics are governed by partial differential equations with local conditions imposed on the boundaries. To not confuse chain structured graphs with chain graphs, in this paper we refer to chain graphs as boundary valued graphs. Such graphical models have been used extensively in the past to model images with boundary conditions being either Dirichlet, Neumann, or periodic, see [20], [26]–[29] for examples. To enable recursive processing, past work has either ignored the effect of boundaries or assumed simpler boundary values. Using our block-tree graph framework, we cluster all boundary nodes in the chain graph into one cluster and then build the block-tree graph. In [30], we derived recursive representations, which we called a telescoping representation, for random fields over continuous indices and random fields over lattices with boundary conditions. The results presented here extend the telescoping

representations to arbitrary boundary valued graphs, not necessarily restricted to boundary valued graphs over 2-D lattices. Applying our block-tree graph framework to Gaussian graphical models, we get linear state-space representations, which leads to recursive estimation equations like the Kalman filter [6] or the Rauch-Tung-Striebel [31] smoother.

As mentioned earlier, the complexity of inference in graphical models is exponential in the treewidth of the graph. Thus, inference in graphical models is computationally intractable when the treewidth is large [32]. For this reason, there is interest in efficient approximate inference algorithms. Loopy belief propagation (LBP), where we ignore the cycles in a graph and apply belief propagation, is a popular approach to approximate inference [3]. Although LBP works well in several graphs, convergence of LBP is not guaranteed, or the convergence rate may be slow [8], [33]. Another class of algorithms is based on decomposing a graph into several computationally tractable subgraphs and using the estimates on the subgraphs to compute the final estimate [8], [34], [35]. We show how block-tree graphs can be used to derive efficient algorithms for estimation in graphical models. The key step is in using the block-tree graph to find subgraphs, which we call *spanning block-trees*. We apply the spanning block-tree framework to the problem of estimation in Gaussian graphical models and show the improved performance over spanning trees.

Organization: Section II reviews graphical models, inference algorithms for graphical models, and the junction-tree algorithm. Section III introduces block-tree graphs, outlines an algorithm for constructing block-tree graphs given an arbitrary undirected graph, and introduces optimal block-tree graphs. Section IV outlines an algorithm for inference over block-tree graphs and discusses the computational complexity of such algorithms. Section V considers the special case of boundary valued graphs. Section VI considers the special case of Gaussian graphical models and derives linear recursive state-space representations, using which we outline an algorithm for recursive estimation in graphical models. Section VII considers the problem of approximate estimation of Gaussian graphical models by computing spanning block-trees. Section VIII summarizes the paper.

II. BACKGROUND AND PRELIMINARIES

Section II-A reviews graphical models. For a more complete study, we refer to [36]. Section II-B reviews inference algorithms for graphical models.

A. Review of Graphical Models

Let $\mathbf{x} = \{x_s \in \mathbb{R}^d : s \in V\}$ be a random vector defined on a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes and $E \subset V \times V$ is the set of edges. Given any subset $W \subset V$, let $x_W = \{x_s : s \in W\}$ denote the set of random variables on W. An edge between two nodes s and t can either be directed, which refers to an edge from node s to node t, or undirected, where the ordering does not matter, i.e., both (s, t) and (t, s) belong to the edge set E. One way of representing the edge set is via an $n \times n$ *adjacency matrix* A such that $A(i, j) = 1$ if $(i, j) \in E$, $A(i, j) = 0$ if $(i, j) \notin E$, where we assume $A(i, i) = 1$ for all $i = 1, \ldots, n$. A *path* is a sequence of nodes such that there is either an undirected or directed edge between any two consecutive nodes in the path. A graph with only directed edges is called a directed graph. A directed graph with no cycles, i.e., there is no path with the same start and end node, is called a directed acyclic graph (DAG). A graph with only undirected edges is called an undirected graph. Since DAGs can be converted to undirected graphs via moralization, see [36], in this paper, unless mentioned otherwise, we only study undirected graphs. For any $s \in V$, $\mathcal{N}(s) = \{t \in V : (s, t) \in E\}$ defines the *neighborhood* of s in the undirected graph $G = (V, E)$. The *degree* of a node s, denoted $d(s)$, is the number of neighbors of s. A set of nodes $\mathcal{C}$ in an undirected graph is a *clique* if all the nodes in $\mathcal{C}$ are connected to each other, i.e., all nodes in $\mathcal{C}$ have an undirected edge. A random vector $\mathbf{x}$ defined on an undirected graph G is referred to as an *undirected graphical model* or a *Markov random field*. The edges in an undirected graph are used to specify a set of conditional independencies in the random vector $\mathbf{x}$. For any disjoint subsets A, B, C of V, we say that B separates A and C if all the paths between A and C pass through B. For undirected graphical models, the global Markov property is defined as follows:

Definition 1 (Global Markov Property): For subsets A, B, C of V such that B separates A and C, x_A is conditionally independent of x_C given x_B, i.e., $x_A \perp x_C | x_B$.

By the Hammersley-Clifford theorem, the probability distribution $p(\mathbf{x})$ of Markov models is factored in terms of cliques as [37]

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C), \tag{1}$$

where $\{\psi_C(x_C)\}_{C \in \mathcal{C}}$ are positive potential functions, also known as clique functions, that depend only on the variables in the clique $C \in \mathcal{C}$, and Z, the partition function, is a normalization constant.

Throughout the paper, we assume that a given graph is connected, which means that there exists a path between any two nodes of the graph. If this condition does not hold, we can always split the graph into more than one connected graph and separately study each connected graph. A *subgraph* of a graph
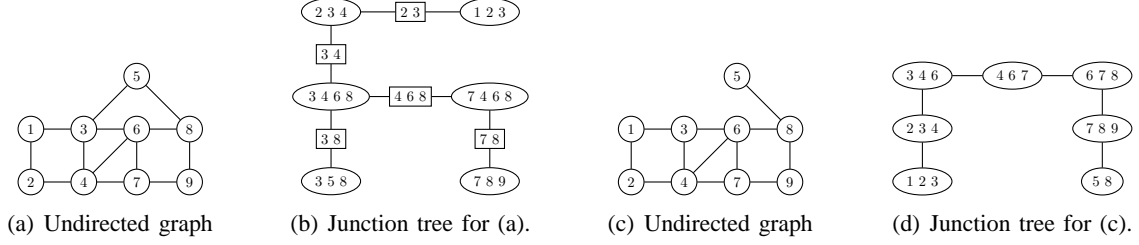
(a) Undirected graph   (b) Junction tree for (a).   (c) Undirected graph   (d) Junction tree for (c).

Fig. 1. Undirected graphs and their junction-trees.

$G = (V, E)$ is graph with vertices and edges being a subset of V and E, respectively. In the next Section, we review algorithms for doing inference in graphical models.

B. Inference Algorithms

Inference in graphical models corresponds to finding marginal distributions, say $p(x_s)$, given the joint probability distribution $p(\mathbf{x})$ for $\mathbf{x} = \{x_1, \ldots, x_n\}$. All inference algorithms derived on $p(\mathbf{x})$ can be applied to the problem of estimation, where we want to marginalize the joint distribution $p(\mathbf{x}|\mathbf{y})$ to find $p(x_s|\mathbf{y})$, where $\mathbf{y}$ is a noisy observation of the random vector $\mathbf{x}$.

For tree-structured graphs, belief propagation [3] is an efficient algorithm for inference with complexity linear in the number of nodes. For graphs with cycles, as discussed in Section I, a popular method is to first construct a junction-tree and then apply belief propagation [12]. We now consider two examples that will act as running examples throughout the paper.

Example 1: Consider the undirected graph in Fig. 1(a) and its junction-tree shown in Fig. 1(b). The clusters in the junction-tree are represented as ellipses (these are the cliques in the triangulated graph producing the junction-tree). On the edges connecting clusters, we have separator nodes that correspond to the common nodes connecting two clusters. It can be shown that this junction-tree is optimal, and thus the treewidth of the graph in Fig. 1(a) is three.

Example 2: By deleting the edge between nodes 3 and 5 in Fig. 1(a), we get the undirected graph in Fig. 1(c). The optimal junction tree is shown in Fig. 1(d) (the separator nodes are ignored for simplicity). The treewidth of the graph is two.

To do inference using junction-trees, we first associate potential functions with each clique. This is done by grouping potentials from the original joint distribution and mapping them to their respective cliques. For example, in Example 1, the junction tree has a clique $\{1, 2, 3\}$, so the clique function will be $\Psi_{1,2,3} = \psi_{1,2}(x_1, x_2)\psi_{1,3}(x_1, x_3)$, where $\psi_{1,2}(x_1, x_2)$ and $\psi_{1,3}(x_1, x_3)$ are factors in the original

probability distribution. Having defined potential functions for each clique, a message passing scheme, similar in spirit to belief propagation, can be formulated to compute marginal distributions for each clique [12], [16]. The marginal distribution of each node can be subsequently computed by marginalizing the distribution of the cliques. The following theorem summarizes the time and space complexity of doing inference on junction trees.

Theorem 1 (Complexity of inference using junction tree [16]): For a random vector $\mathbf{x} \in \mathbb{R}^n$ defined on an undirected graph $G = (V, E)$ with each x_s taking values in Ω, the time complexity for doing inference is exponential in the treewidth of the graph and the space complexity of doing inference is exponential in the treewidth of the graph plus one.

Theorem 1 corresponds to the complexity of doing inference on an optimal junction tree. However, finding the optimal junction-tree is hard in general, and thus the complexity is estimated by the upper bound of the treewidth of the graph, which can be found using algorithms in [18]. The next Section introduces block-tree graphs as an alternative tree decomposition for graphs with cycles and shows that constructing block-tree graphs is less computationally intensive than constructing junction-trees and finding optimal block-trees has a smaller search space than finding optimal junction-trees.

III. BLOCK-TREE GRAPH

In this section, we introduce block-tree graphs and show the merits of using block-tree graphs over junction-trees. Section III-A defines a block-tree graph and gives examples. Section III-B shows how to construct block-tree graphs starting from a connected undirected graph. Section III-C introduces optimal block-tree graphs.

A. Definition and Examples

To define a block-tree graph, we first introduce block-graphs, which generalize the notion of graphs. Throughout this paper, we denote block-graphs by $\mathcal{G}$ and graphs by G.

Definition 2 (Block-graph): A *block-graph* is the tuple $\mathcal{G} = (\mathcal{C}, \mathcal{E})$, where $\mathcal{C} = \{C_1, C_2, \ldots, C_l\}$ is a set of disjoint clusters and $\mathcal{E}$ is a set of edges such that $(i, j) \in \mathcal{E}$ if there exists an edge between the clusters C_i and C_j.

Let the cardinality of each cluster be $\gamma_k = |C_k|$, and let n be the total number of nodes. If $\gamma_k = 1$ for all k, $\mathcal{G}$ reduces to an undirected graph. For every block-graph $\mathcal{G}$, we associate a graph $G = (V, E)$, where V is the set of all nodes in the graph and E is the set of edges between nodes of the graph. For

(a) Block-tree   (b) An undirected graph for (a)   (c) Junction tree for Fig. 2(b)   (d) Block-tree for Fig. 1(c)
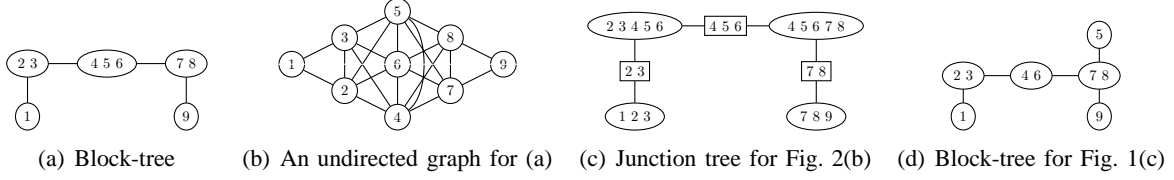
Fig. 2.   Example of block-trees

each $(i, j) \in \mathcal{E}$, the set of edges E will contain at least one edge connecting two nodes in C_i and C_j or connecting nodes within C_i or C_j. A complete block-graph is defined as follows.

Definition 3 (Complete block-graph): For a block-graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$, if all the nodes in C_i have an edge between them, and for all $(i, j) \in E$ if all the nodes in C_i and all the nodes in C_j have an edge between them, then $\mathcal{G}$ is a complete block graph.

We now introduce block-tree graphs.

Definition 4 (Block-tree graph): A block-graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ is called a block-tree graph if there exists only one path connecting any two clusters C_i and C_j.

Thus, block-tree graphs generalize tree-structured graphs. Using Definition 3, we can define a complete block-tree graph. As an example, consider the block-tree graph shown in Fig. 2(a), where $C_1 = \{1\}$, $C_2 = \{2, 3\}$, $C_3 = \{4, 5, 6\}$, $C_4 = \{7, 8\}$, and $C_5 = \{9\}$. A complete block-tree graph corresponding to the block-tree in Fig. 2(a) is shown in Fig. 2(b). The block-tree graph in Fig. 2(a) serves as a representation for a family of undirected graphs. For example, Fig. 2(a) serves as a representation for the graphs in Fig. 1(a) and Fig. 1(c). This can be seen by removing edges from Fig. 2(b). In the next Section, we consider the problem of constructing a block-tree graph given an undirected graph.

B. Constructing Block-Tree Graphs

Our algorithm for constructing block-tree graphs is outlined in Algorithm 1. The input to the algorithm is a connected graph G and an initial cluster V_1, which we call the *root cluster*. The output of the algorithm is a block-tree graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$. The key steps of the algorithm are highlighted as follows:

Forward Pass: (Lines 3-7) Starting from the root cluster V_1, we iteratively find successive neighbors of V_1 to construct a sequence of r clusters $V_1, V_2, \ldots, V_r$ such that $V_2 = \mathcal{N}(V_1) \backslash V_1$, $V_3 = \mathcal{N}(V_2) \backslash \{V_1 \cup V_2\}$, $\ldots$, $V_r = \mathcal{N}(V_{r-1}) \backslash \{V_{r-2} \cup V_{r-1}\}$, where $\mathcal{N}(V_k)$ are the neighbors of the set of nodes in V_k. This is shown in Line 5 of Algorithm 1. For each V_k, $k = 2, \ldots, r$, we split V_k into m_k disjoint clusters $\{V_k^1, \ldots, V_k^{m_k}\}$ such that $\bigcup V_k^j = V_k$ and there are no edges between the clusters V_1^i and V_1^j for $i \neq j$. This is shown in Line 6 of Algorithm 1.

Algorithm 1 Constructing Block-tree Graphs

1: **procedure** CONSTRUCTBLOCKTREE(G, V_1)
2:     $r = 1$; $V_0 = \{\ \}$;
3:     **while** $\bigcup V_r \neq V$ **do**
4:         $r = r + 1$
5:         Find neighbors: $V_r = \{k : (j, k) \in E \ \forall \ , j \in V_{r-1}\} \backslash \{V_{r-2} \cup V_{r-1}\}$
6:         Split Cluster: $\{V_r^1 \dots, V_r^{m_r}\}$ s.t. for all $i \in V_r^{n_1}$ and $j \in V_r^{n_2}$, $n_1 \neq n_2$, $(i, j) \notin E$.
7:     **end while**
8:     **for** $i = r, r-1, \dots, 3$ **do**
9:         **for** $j = 1, \dots, m_i$ **do**
10:             Update cluster: Find $\{V_{i-1}^{j_1}, \dots, V_{i-1}^{j_w}\}$ s.t. there exists nodes $s_1, \dots, s_w$, where $s_k \in V_{i-1}^{j_k}$,
    s.t. $(s_k, t) \in E$ for some $t \in V_i^j$. Combine $\{V_{i-1}^{j_1}, \dots, V_{i-1}^{j_w}\}$ into one cluster and update V_{i-1}.
11:         **end for**
12:     **end for**
13:     Relabel clusters as $C_1, \dots, C_l$ and find edge set $\mathcal{E}$ s.t. $(i, j) \in \mathcal{E}$ if there exists an edge between
    C_i and C_j.
14: **end procedure**

Backwards Pass: (Lines 8-13) In this step, we find the final clusters of nodes given $V_1, V_2, \dots, V_r$. Starting at $V_r = \{V_r^1, \dots, V_r^{m_r}\}$, for each V_r^j, $j = 1, \dots, m_r$, we find all clusters $\{V_{r-1}^{j_1}, \dots, V_{r-1}^{j_w}\}$ such that there exists an edge between $V_{r-1}^{j_n}$, $n = 1, \dots, w$ and V_r^j. Combine $\{V_{r-1}^{j_1}, \dots, V_{r-1}^{j_w}\}$ into one cluster and then update the clusters in V_{r-1} accordingly. We repeat the above steps for all clusters $V_{r-1}, \dots, V_3$. Thus, if $r = 2$, the backwards step is not needed. Relabel all the clusters such that $\mathcal{C} = \{C_1, \dots, C_l\}$ and find the edge set $\mathcal{E}$.

The forward pass of the algorithm first finds a chain structured block-graph over the clusters $V_1, V_2, \dots, V_r$. The backwards pass then splits the clusters in V_k to get a tree-structured graph. The key intuition utilized in the backwards pass is that each cluster in V_k connects to only one cluster in V_{k-1}. If there are more than one such clusters in V_{k-1}, it is trivial to see that the resultant block-graph will have a cycle and will no longer be a block-tree graph.

As an example, consider finding block-tree graphs for the graph in Fig. 1(a). Starting with the root cluster $V_1 = \{1\}$, we have $V_2 = \{2, 3\}$, $V_3 = \{4, 5, 6\}$, $V_4 = \{7, 8\}$, and $V_5 = \{9\}$. Further splitting the clusters V_k and running the backwards pass, the clusters do not split, and we get the block-tree graph in Fig. 2(a). We get the same block-tree graph if we start from the root cluster $V_1 = \{9\}$. Now suppose, we start with the root cluster $V_1 = \{2, 3\}$. Then, we have $V_2 = \{1, 4, 5, 6\}$, $V_3 = \{7, 8\}$, and $V_4 = \{9\}$. Splitting these clusters (Line 6 in Algorithm 1), we have $V_2^1 = \{1\}$, $V_2^2 = \{4, 6\}$, $V_2^3 = \{5\}$, $V_3^1 = \{7\}$, $V_3^2 = \{8\}$, and $V_4^1 = \{4\}$. Given these clusters, we now apply the backwards pass to find the final set

of clusters:

1) The cluster $V_4^1 = \{4\}$ has edges in both $V_3^1 = \{7\}$ and $V_3^2 = \{8\}$, so we combine V_3^1 and V_3^2 to get $V_3^1 = \{7, 8\}$.

2) Next, the cluster $V_3^1 = \{7, 8\}$ has edges in V_2^2 and V_2^3, so we combine these clusters and get $V_2^1 = \{1\}$ and $V_2^2 = \{4, 5, 6\}$.

Given the above clusters, we get the same block-tree graph in Fig. 2(a). The need of the backwards pass in Algorithm 1 is clear from the above example since it successfully splits the cluster V_2 with four nodes into two smaller clusters. As another example, the block-tree graph for the graph in Fig. 1(c) using a root cluster of $V_1 = \{1\}$ is shown in Fig. 2(d).

Notice that in Algorithm 1 we did not split the root cluster V_1. Thus, one of the clusters in $\mathcal{C}$ will be V_1. Without loss in generality, we assume $C_1 = V_1$. We now show that Algorithm 1 always gives us a unique block-tree graph for each set of root cluster V_1.

Theorem 2: Algorithm 1 always outputs a block-tree graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ for each possible set of root cluster V_1 and undirected graph $G = (V, E)$, which is connected. Further, the block-tree graph $\mathcal{G}$ is unique.

Proof: For the root cluster V_1, after the backwards pass of the algorithm, we have the set of clusters: $V_1, \{V_2^1, V_2^2, \ldots, V_2^{m_2}\}, \ldots, \{V_r^1, V_r^2, \ldots, V_r^{m_r}\}$. By construction, there are no edges between $V_k^{n_1}$ and $V_k^{n_2}$ for $n_1 \neq n_2$. The total number of clusters is $l = 1 + \sum_{k=2}^r m_k$. For $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ to be a block-tree graph, the undirected graph $G = (\{1, 2, \ldots, l\}, \mathcal{E})$ to be a tree-structured graph. For this, G must be connected and the number of edges in the graph must be $|\mathcal{E}| = l - 1$. The block-tree graph formed using Algorithm 1 is connected by construction since the original undirected graph $G = (V, E)$ is connected. Counting the number of edges between clusters, we have $l - 1$ edges, and thus the output of Algorithm 1 is a block-tree graph. The uniqueness of the block-tree graph follows from construction. ∎

The next theorem characterizes the complexity of constructing block-tree graphs.

Theorem 3 (Complexity of Algorithm 1): The complexity of constructing a block-tree graph is $O(n)$, where n is the number of nodes in the graph.

Proof: The proof is trivial since the algorithm involves traversing the nodes of the graph. We do this twice, once during the forward pass and once during the backwards pass. ∎

Comparison to junction-trees: As mentioned before, the key difference between block-trees and junction-trees is that block-trees are constructed using disjoint clusters, whereas clusters in a junction-tree have common nodes. Constructing block-trees is computationally more efficient since constructing junction-trees requires an additional complexity of $O(m^2)$, where m can be as large as n for sparse graphs. From

Algorithm 1 and Theorem 2, we note that a block-tree graph is uniquely specified using a root cluster, which is a small number of nodes. On the other hand, specifying a junction-tree requires an elimination order, the size of which can be as large[2] as n.

C. Optimal Block-Tree Graphs

In this Section, we consider the problem of finding optimal block-tree graphs. In order to define an optimal block-tree graph, we first introduce the notion of block-width and block-treewidth.

Definition 5 (Block-width): For an undirected graph $G = (V, E)$, the *block-width* of the graph with respect to a root cluster V_1, $\text{bw}(G, V_1)$, is the maximum cluster size in the block-tree graph constructed using V_1 as the root cluster: $\text{bw}(G, V_1) = \max_k |\gamma_k|$.

Definition 6 (Block-treewidth): The *block-treewidth*, $\text{btw}(G)$, of an undirected graph $G = (V, E)$ is the minimal block-width of a graph G with respect to all root clusters: $\text{btw}(G) = \min_{V_1 \subset V} \text{bw}(G, V_1)$.

For example, $\text{bw}(G, \{1\}) = 3$ for the graph in Fig. 1(a). By checking over all possible root clusters, it is easy to see that the block-treewidth for the graph in Fig. 1(a) is also three. For Fig. 1(c), $\text{bw}(G, \{1\}) = 2$ which is also the block-treewidth of the graph. We can now define an optimal block-tree graph.

Definition 7 (Optimal block-tree graph): A block-tree graph for an undirected graph $G = (V, E)$ with respect to a root cluster V_1 is optimal if the block-width with respect to V_1 is equal to the block-treewidth of the graph, i.e., $\text{bw}(G, V_1) = \text{btw}(G)$.

We show in Section VII-A that the notion of optimality for block-tree graphs in Definition 7 is useful when finding spanning block-trees, which are subgraphs with lower block-treewidth. Computing the block-treewidth of a graph requires a search over all possible root clusters, which has complexity of $O(2^n)$. This search space can be simplified since if we choose a V_1 such that $|V_1| \geq \lceil n/2 \rceil$, the block-width of the graph will be V_1 itself. Thus, the search space can be restricted to root clusters of length $\lceil n/2 \rceil$, which requires a search over $\binom{n}{n/2}$ number of possible clusters. In comparison, computing the treewidth requires finding an optimal elimination order, which requires a search over $n!$ possible number of combinations. Since $n! \gg \binom{n}{\lceil n/2 \rceil}$, the search space of computing the treewidth is much larger than the search space of computing the block-treewidth. However, the problem of computing the block-treewidth is still computationally intractable as the search space grows exponentially as n increases.

We now propose a simple heuristic to find an upper bound on the block-treewidth of a graph. Let $G = (V, E)$ be an undirected graph with n nodes. Instead of searching over all possible root clusters,

---

[2]The exact size of the elimination order depends on the connectivity of the graph

TABLE I
UPPER BOUND ON BLOCK-TREEWIDTH VS UPPER BOUND ON TREEWIDTH

Graph	Treewidth	Block-treewidth	nodes	edges
ship-ship-pp	8	8	30	77
water	10	8	32	123
fungiuk	4	4	15	36
pathfinder-pp	7	6	12	43
1b67	17	16	68	559
1bbz	28	23	57	543
1bkb	34	29	131	1485
1bkf	39	37	106	1264
1bx7	11	11	41	195
1en2	17	16	69	463
1on2	40	34	135	1527
$n \times n$ grid graph	n	n	n^2	$2n(n-1)$

whose maximum size can be $\lceil n/2 \rceil$, we restrict the search space to smaller root clusters. For n small, we find the best root cluster of size two, and for n large we find the best root cluster of size one. Given the initial choice of the root cluster, we add nodes to this to see if the block-width can be lowered further. For example, if the initial root cluster is V_1, we check over all $k \in V \backslash V_1$ and see if $\{V_1, k\}$ leads to a lower block-width. We repeat this process until the block-width does not decrease further. For small n, the complexity of this heuristic algorithm is $O(n^2)$, since we initially search over all clusters of size two. For large n, the complexity is $O(n)$ since we search only over clusters of size one. Table I compares upper bounds on the treewidth vs. upper bounds on the block-treewidth for some standard graphs used in the literature[3]. The upper bound on the treewidth is computed using a software package[4].

IV. INFERENCE USING BLOCK-TREE GRAPHS

In this Section, we outline an algorithm for inference in undirected graphical models using block-tree graphs. The algorithm is similar to belief propagation with the difference that message passing happens between clusters of nodes instead of individual nodes. Let $\mathbf{x} \in \mathbb{R}^n$ be a random vector defined on an undirected graph $G = (V, E)$. Using V_1 as a root cluster, suppose we construct the block-tree graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ using Algorithm 1, where $\mathcal{C} = [C_1, C_2, \ldots, C_l]$ and $\gamma_k = |C_k|$. From (1), we know that $p(\mathbf{x})$

[3]The graphs were obtained from the database in people.cs.uu.nl/hansb/treewidthlib/

[4]See www.treewidth.com

admits the factorization over cliques such that

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C), \tag{2}$$

where $\mathcal{C}$ is the set of cliques. Using the block-tree graph, we can express the factorization of $p(\mathbf{x})$ as

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{(i,j) \in \mathcal{E}} \Psi_{i,j}\left(x_{C_i}, x_{C_j}\right), \tag{3}$$

where the factors $\Psi_{i,j}\left(x_{C_i}, x_{C_j}\right)$ correspond to a product of potential functions taken from the factorization in (2), where each $\psi_C(x_C)$ is mapped to a unique $\Psi_{i,j}\left(x_{C_i}, x_{C_j}\right)$.

As an example, consider a random vector $\mathbf{x} \in \mathbb{R}^9$ defined on the graphical model in Fig. 1(c). The block-tree graph is given in Fig. 2(d) such that $C_1 = \{1\}$, $C_2 = \{2,3\}$, $C_3 = \{4,6\}$, $C_4 = \{7,8\}$, $C_5 = \{5\}$, and $C_6 = \{9\}$. Using Fig. 1(c), the joint probability distribution can be written as

$$p(\mathbf{x}) = \psi_{1,2}\psi_{1,3}\psi_{2,4}\psi_{3,4}\psi_{3,6}\psi_{4,6}\psi_{4,7}\psi_{6,7}\psi_{6,8}\psi_{7,9}\psi_{8,9}\psi_{8,5}, \tag{4}$$

where we simplify $\psi_{i,j}(x_i, x_j)$ as $\psi_{i,j}$. We can rewrite (4) in terms of the block-tree graph as

$$p(\mathbf{x}) = \Psi_{1,2}(x_{C_1}, x_{C_2})\Psi_{2,3}(x_{C_2}, x_{C_3})\Psi_{3,4}(x_{C_3}, x_{C_4})\Psi_{4,5}(x_{C_4}, x_{C_5})\Psi_{4,6}(x_{C_4}, x_{C_6}),$$

where $\Psi_{1,2}(x_{C_1}, x_{C_2}) = \psi_{1,3}\psi_{1,2}$, $\Psi_{2,3}(x_{C_2}, x_{C_3}) = \psi_{3,6}\psi_{2,4}\psi_{4,6}$, $\Psi_{2,4}(x_{C_3}, x_{C_4}) = \psi_{6,8}\psi_{4,7}\psi_{6,7}$, $\Psi_{4,5}(x_{C_4}, x_{C_5}) = \psi_{8,5}$, and $\Psi_{4,6}(x_{C_4}, x_{C_6}) = \psi_{7,9}\psi_{8,9}$. Since the block-tree graph is a tree decomposition, all algorithms valid for tree-structured graphs can be directly applied to block-tree graphs. Thus, we can now use the belief propagation algorithm discussed in Section II-B to do inference on block-tree graphs. The steps involved are similar, with an additional step to marginalize the joint distributions over each cluster:

1) For any cluster, say C_1, identify its leaves.

2) Starting from the leaves, pass messages along each edge until we reach the root cluster C_1:

$$m_{i \to j}(x_{C_j}) = \sum_{x_{C_i}} \Psi_{i,j}\left(x_{C_i}, x_{C_j}\right) \prod_{e \in \mathcal{N}(i)\backslash j} m_{e \to i}(x_{C_i}), \tag{5}$$

where $\mathcal{N}(i)$ is the neighboring cluster of C_i and $m_{i \to j}(x_{C_j})$ is the message passed from cluster C_i to C_j.

3) Once all messages have been communicated from the leaves to the root, pass messages from the root back to the leaves.

4) After the messages reach the leaves, the joint distribution for each cluster is given as

$$p(x_{C_i}) = \prod_{j \in \mathcal{N}(i)} m_{j \to i}(x_{C_i}).$$ (6)

5) To find the distribution of each node, marginalize $p(x_{C_i})$.

We now analyze the complexity of doing inference using block-tree graphs. Assume $x_s \in \Omega$, where $s \in V$ such that $|\Omega| = K$. From (5), to pass a message from C_i to C_j, for each $x_{C_j} \in \Omega^{|C_j|}$, we require $K^{|C_i|}$ number of additions. Thus, this step will require $K^{|C_i|+|C_j|}$ number of additions since we need to compute $m_{i \to j}(x_{C_j})$ for all possible values x_{C_j} takes. Thus, the complexity of inference is given as follows:

Theorem 4 (Complexity of inference using block-tree graphs): For a random vector $\mathbf{x} \in \mathbb{R}^n$ defined on an undirected graph $G = (V, E)$ with each x_s taking values in Ω, the complexity of performing inference is exponential in the maximum sum of cluster sizes of adjacent clusters.

Another way to realize Theorem 4 is to form a junction-tree using the block-tree graph. It is clear that, in general, using block-tree graphs for inference is computationally less efficient than using junction-trees. However, for complete block-graphs, an example of which is shown in Fig. 2(b), we see that both the junction-tree and the block-tree graph have the same computational complexity. Thus, complete block-graphs are attractive graphical models to use the block-tree graph framework when the goal is to do exact inference. In Section VII, we illustrate the advantage of using the block-tree graph framework on arbitrary graphical models in the context of approximate inference in graphical models.

V. BOUNDARY VALUED GRAPHS

In this Section, we specialize our block-tree graph framework to boundary valued graphs, which are known as chain graphs in the literature. We are only concerned with boundary valued graphs where we have one set of boundary nodes connected in a directed manner to nodes in an undirected graph. The motivation for using these particular types of boundary valued graphs is in modeling physical phenomena whose underlying statistics are governed by partial differential equations satisfying boundary conditions [29], [38]. A common example is in texture modeling, where the boundary edges are often assumed to satisfy either periodic, Neumann, or Dirichlet boundary conditions [20]. If the boundary values are zero, the graph will become undirected, and we can then use the block-tree graph framework in Section III.

Let $G = \left(\{V, \partial V\}, E_V^- \cup E_{\partial V}^- \cup E^{\to}\right)$ be a boundary valued graph, where V^- is a set of nodes, called *interior nodes*, and ∂V is another set of nodes referred to as *boundary nodes*. We assume that the nodes
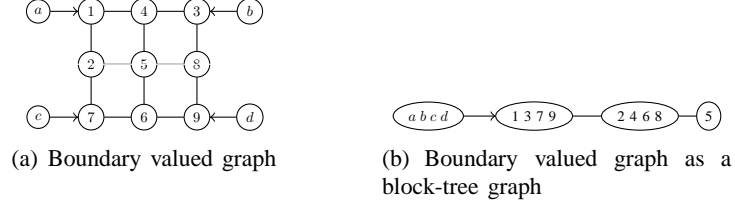
(a) Boundary valued graph

(b) Boundary valued graph as a block-tree graph

Fig. 3. An example of a boundary valued graph

in V^- and ∂V are connected by undirected edges, denoted by the edge sets E_V^- and $E_{\partial V}^-$, and that there exist directed edges between the nodes of V^- and ∂V. To construct a block-tree graph, we first need to chose a cluster C_1. As discussed in Section III, any choice of clusters can lead to a recursive representation; however, it is natural for boundary valued graphs to initiate at the boundary. For this reason, we let the root cluster be ∂V and then use Algorithm 1 to construct a block-tree graph. By choosing the boundary values first, we convert a boundary valued problem into an initial valued problem.

An example of a boundary valued graph and its block-tree graph is shown in Fig. 3. We let C_1 be the boundary nodes $\{a, b, c, d\}$ and subsequently construct $C_2 = \{1, 3, 7, 9\}$, $C_3 = \{2, 4, 6, 8\}$, and $C_4 = \{5\}$ so that we get the chain structured graph in Fig. 3(b). The probability distribution of $\mathbf{x}$ defined on this graph can be written as

$$p(\mathbf{x}) = P(a)P(b)P(c)P(d)\psi_{abcd}\psi_{1,a}\psi_{3,b}\psi_{7,c}\psi_{9,d}\psi_{1:9} \text{, where,} \tag{7}$$

$$\psi_{1:9} = \psi_{12}\psi_{14}\psi_{2,5}\psi_{2,3}\psi_{3,6}\psi_{5,6}\psi_{4,5}\psi_{6,9}\psi_{8,9}\psi_{5,8}\psi_{7,8}\psi_{4,7} \,,$$

$$\psi_{abcd}^{-1} = \sum_{1:9} \psi_{1,a}\psi_{3,b}\psi_{7,c}\psi_{9,d}\psi_{1:9} \,. \tag{8}$$

Using the block-tree graph, we write the probability distribution as

$$p(\mathbf{x}) = \Psi_{1,2}(x_{V_1}, x_{V_2})\Psi_{2,3}(x_{V_2}, x_{V_3})\Psi_{3,4}(x_{V_3}, x_{V_3}) \text{, where,} \tag{9}$$

$$\Psi_{1,2}(x_{V_1}, x_{V_2}) = P(a)P(b)P(c)P(d)\psi_{abcd}\psi_{1,a}\psi_{3,b}\psi_{7,c}\psi_{9,d} \tag{10}$$

$$\Psi_{2,3}(x_{V_2}, x_{V_3}) = \psi_{12}\psi_{1,4}\psi_{3,2}\psi_{3,6}\psi_{7,4}\psi_{7,8}\psi_{6,9}\psi_{8,9} \tag{11}$$

$$\Psi_{3,4}(x_{V_3}, x_{V_3}) = \psi_{2,5}\psi_{4,5}\psi_{6,5}\psi_{8,5} \tag{12}$$

Notice that (9) did not require the calculation of a normalization constant Z. This calculation is hidden in the potential function ψ_{abcd} given by (8). We note that the results presented here extend our results for deriving recursive representations for Gaussian lattice models with boundary conditions in [30] to arbitrary (non-Gaussian) undirected graphs with boundary values.
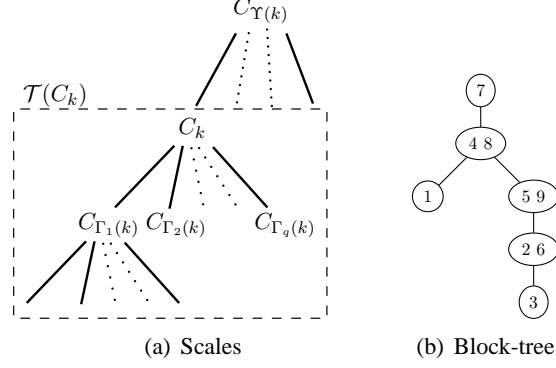
(a) Scales   (b) Block-tree

Fig. 4. Block-tree graph and related shift operators.

VI. GAUSSIAN GRAPHICAL MODELS

In this Section, we specialize our block-tree graph framework to Gaussian graphical models. Section VI-A reviews Gaussian graphical models and introduces relevant notations. Section VI-B derives linear state-space representations for undirected Gaussian graphical models. Using these state-space representations, we derive recursive estimators in Section VI-C.

A. Preliminaries

Let $\mathbf{x} \in \mathbb{R}^n$ be a Gaussian random vector defined on a graph $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, and[5] $x_k \in \mathbb{R}$. Without loss in generality, we assume that $\mathbf{x}$ has zero mean and covariance Σ. From [1], it is well known that the inverse of the covariance is sparse and the nonzero patterns in $J = \Sigma^{-1}$ determine the edges of the graph G. In the literature, J is often referred to as the information matrix or the potential matrix.

Suppose we construct a block-tree graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ from the undirected graph $G = (V, E)$. Let P be a permutation matrix which maps V to $\mathcal{C} = \{C_1, C_2, \ldots, C_l\}$, i.e.,

$$x(\mathcal{C}) = Px(V) = P\mathbf{x}.$$

The covariance of $x(\mathcal{C})$ is

$$E[x(\mathcal{C})x(\mathcal{C})^T] = P\Sigma P^T = [\Sigma_P(i,j)]_l \tag{13}$$

$$E[x(C_i)x(C_j)^T] = \Sigma_P(i,j), \text{ for } i, j = 1, \ldots, l, \tag{14}$$

---

[5]For simplicity, we assume $x_k \in \mathbb{R}$, however, our results can be easily generalized when $x_k \in \mathbb{R}^d$, $d \geq 2$.

where the notation $\mathbf{F} = [F(i,j)]_l$ refers to the blocks of the matrix $\mathbf{F}$ for $i, j = 1, \ldots, l$. We now define some notation that will be useful in deriving the state-space representation and the recursive estimators. The notation is borrowed from standard notation used for tree-structured graphs in [4], [5], [39]. For the block-tree graph, we have defined C_1 as the root cluster. The other clusters of the block-tree graph can be partially ordered according to their *scale*, which is the distance of a cluster to the root cluster C_1. This distance is defined as the number of edges in the path connecting two clusters. Since we have a tree graph over the clusters, a path connecting two clusters is unique. Thus, the scale of C_1 is zero. All the neighbors of C_1 will have scale one. For any C_k at scale s, define $\{C_{\Gamma_1(k)}, \ldots, C_{\Gamma_q(k)}\}$ as the set of clusters connected to C_k at scale $s + 1$ and $C_{\Upsilon(k)}$ as the cluster connected to C_k at scale $s - 1$. Let $\mathcal{T}(C_k)$ be all the clusters including C_k at scale greater than s and at a distance greater than zero:

$$\mathcal{T}(C_k) = \{C_i : d(C_k, C_i) \geq 0 \text{ and } \text{scale}(C_i) > s\}, \tag{15}$$

where $\text{scale}(C_i)$ is the scale of the cluster C_i. Fig. 4(a) shows the relevant notations on a block-tree graph. Fig. 4(b) shows an example of a block-tree graph where the root cluster is $C_1 = \{7\}$, the cluster at scale one is $C_2 = \{7, 8\}$, the clusters at scale two are $C_3 = \{1\}$ and $C_4 = \{5, 9\}$, the cluster at scale three is $C_5 = \{2, 6\}$, and the cluster at scale four is $C_6 = \{3\}$. For C_2, we have that $C_{\Gamma_1(2)} = C_3$, $C_{\Gamma_2(2)} = C_4$, and $C_{\Upsilon(2)} = C_1$. In the next Section, we derive a state-space representation for Gaussian random vectors defined on undirected graphs using block-tree graphs.

B. State-Space Representation

Given a block-tree graph, we now derive state-space representations on the tree structure. There are two types of representations we can define, one in which we are given $x(C_k)$ at scale s, and we want to compute $x(C_{\Upsilon(k)})$ at scale $s - 1$, and another in which we are given $x(C_{\Upsilon(k)})$ at scale $s - 1$, and we want to compute $x(C_k)$ at scale s.

Theorem 5: Let $\mathbf{x} = \{x_k \in \mathbb{R} : k \in V\}$ be a random vector defined over a graph $G = (V, E)$ and let $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ be a block-tree graph obtained using a root cluster C_1. We have the following linear representations:

$$x(C_k) = A_k x(C_{\Upsilon(k)}) + u(C_k) \tag{16}$$
$$x(C_{\Upsilon(k)}) = F_k x(C_k) + w(C_k), \tag{17}$$

where $u(C_k)$ is a *white* Gaussian noise uncorrelated with $x(C_{\Upsilon(k)})$, $w(C_k)$ is *non-white* Gaussian noise uncorrelated with $x(C_k)$, and

$$A_k = \Sigma_P(C_k, C_{\Upsilon(k)}) \left[\Sigma_P(C_{\Upsilon(k)}, C_{\Upsilon(k)})\right]^{-1} \tag{18}$$

$$Q_k^u = E\left[u(C_k)u^T(C_k)\right] = \Sigma_P(C_k, C_k) - A_k\Sigma_P(C_{\Upsilon(k)}, C_k) \tag{19}$$

$$F_k = \Sigma_P(C_{\Upsilon(k)}, C_k) \left[\Sigma_P(C_k, C_k)\right]^{-1} \tag{20}$$

$$Q_k^w = E\left[w(C_k)w^T(C_k)\right] = \Sigma_P(C_{\Upsilon(k)}, C_{\Upsilon(k)}) - F_k\Sigma_P(C_k, C_{\Upsilon(k)}), \tag{21}$$

where $\Sigma_P(i,j)$ is a block from the covariance of $x(\mathcal{C})$ defined in (14).

Proof: We first derive (16). Consider the cluster of nodes $\mathcal{C}\backslash\mathcal{T}(C_k)$. From the global Markov property, stated in Definition 1, we have

$$\widehat{x}(C_k) = E\left[x(C_k)|\{x(C_i) : C_i \in \mathcal{C}\backslash\mathcal{T}(C_k)\}\right] = E\left[x(C_k)|x(C_{\Upsilon(k)})\right]. \tag{22}$$

$$= \Sigma_P(C_k, C_{\Upsilon(k)}) \left[\Sigma_P(C_{\Upsilon(k)}, C_{\Upsilon(k)})\right]^{-1} x(C_{\Upsilon(k)}) = A_k x(C_{\Upsilon(k)}), \tag{23}$$

where we get (23) using the Gauss-Markov theorem [40] for computing the conditional mean. Define the error $u(C_k)$ as

$$u(C_k) = x(C_k) - \widehat{x}(C_k) = x(C_k) - A_k x(C_{\Upsilon(k)}). \tag{24}$$

It is clear that $u(C_k)$ is Gaussian. Further, by the orthogonality properties of the minimum-mean squared error (mmse) estimates, $u(C_k)$ is white. The variance of $u(C_k)$ is computed as follows:

$$E\left[u(C_k)u^T(C_k)\right] = E\left[(x(C_k) - \widehat{x}(C_k))(x(C_k) - \widehat{x}(C_k))^T\right] \tag{25}$$

$$= E\left[(x(C_k) - \widehat{x}(C_k))x^T(C_k)\right] \tag{26}$$

$$= \Sigma_P(C_k, C_k) - A_k\Sigma_P(C_{\Upsilon(k)}, C_k). \tag{27}$$

To go from (25) to (26), we use the orthogonality of $u(C_k)$. This gives us the Q_k^u in (19). Equation (17) can be either derived in a similar manner or alternatively by using the results of [41] on backwards Markovian models. ∎

The driving noise in (17), $w(C_k)$, is not white noise. This happens because for each $C_{\Upsilon(k)}$, there can be more than one cluster such that $C_{\Upsilon(j)} = C_{\Upsilon(k)}$. Using the state-space representations, we can easily recover standard recursive algorithms, an example of which is shown in the next Section where we consider the problem of estimation over Gaussian graphical models.

C. Recursive Estimation

Let $\mathbf{x} \in \mathbb{R}^n$ be a zero mean Gaussian random vector defined on an undirected graph $G = (V, E)$. Let Σ be the covariance of $\mathbf{x}$ and let $J = \Sigma^{-1}$. Suppose we collect noisy observations of $\mathbf{x}$ such that

$$y_s = H_s x_s + n_s \,, \tag{28}$$

where $n_s \sim \mathcal{N}(0, R_s)$ is white Gaussian noise independent of x_s and H_s is known. Given $\mathbf{y} = [y_1, \ldots, y_n]^T$, we want to find the minimum-mean squared error (mmse) estimate of $\mathbf{x}$, which is $E[\mathbf{x}|\mathbf{y}]$. From the Gauss-Markov theorem [40], we have

$$\widehat{x} = E[\mathbf{x}|\mathbf{y}] = E[\mathbf{x}\mathbf{y}^T] \left(E[\mathbf{y}\mathbf{y}^T] \right)^{-1} \mathbf{y} \tag{29}$$

$$= \Sigma \left(H\Sigma H^T + R \right)^{-1} \mathbf{y} \,, \tag{30}$$

where H and R are diagonal matrices with diagonals H_s and R_s, respectively. Using (30) to compute $\widehat{\mathbf{x}}$ requires inversion of a $n \times n$ matrix, which has complexity $O(n^3)$. An alternate method is to use the state-space representations in Theorem 5 and derive standard Kalman filters and recursive smoothers using [4], [5] [6]. This approach will require inversion of a $\text{btw}(G) \times \text{btw}(G)$ matrix, where $\text{btw}(G)$ is the treewidth of the graph. Another approach to computing $\widehat{\mathbf{x}}$ is to use the equations in [42], where they derive estimation equations for Gaussian tree distributions given $J = \Sigma^{-1}$. The generalization to block-tree graphs is trivial and will only involve identifying appropriate blocks from the inverse of the covariance matrix. Thus, by converting an arbitrary graph into a block-tree graph, we are able to recover algorithms for recursive estimation of Gaussian graphical models. For graphs with high block-treewidth, however, computing mmse estimates is computationally intractable. For this reason, we propose an efficient approximate estimation algorithm in the next Section.

VII. APPROXIMATE ESTIMATION

In this Section, we use the block-tree graph framework to derive approximate estimation algorithms for Gaussian graphical models. The need for approximate estimation arises because estimation/inference in graphical models is computationally intractable for graphical models with large treewidth or large block-treewidth. The approach we use for approximate estimation is based on decomposing the original graphical model into computationally tractable subgraphs and using the subgraphs for estimation, see [8], [34], [35] for a general class of algorithms. Traditional approaches to finding subgraphs involve using

---

[6]The results in [4] are on dyadic trees, however they can be easily generalized to arbitrary trees.

spanning trees, which are tree-structured subgraphs. We propose to use *spanning block-trees*, which are block-tree graphs with low block-treewidth. Section VII-A outlines a heuristic algorithm for finding maximum weight spanning block-trees. We review the matrix splitting approach to approximate estimation in Section VII-B and show how spanning block-trees can be used instead of spanning trees. Section VII-C shows experimental results.

A. Spanning Block-Trees

Let $G = (V, E)$ be an undirected graph. We define a B-width spanning block-tree as a subgraph of G with block-treewidth at most B. If $B = 1$, the spanning block-tree becomes a spanning tree. For $B > 1$, we want to remove edges from the graph G until we get a block-tree graph with block-width less than or equal to B. To quantify each edge, we associate a weight $w_{i,j}$ for each $(i, j) \in E$. If $w_{i,j}$ is the mutual information between nodes i and j, finding an optimal spanning block-tree by removing edges reduces to minimizing the Bethe free energy [43], [44]. For the purpose of approximate estimation in Gaussian graphs, the authors in [45] proposed weights that provided a measure of error-reduction capacity of each edge in the graph. We use these weights when finding spanning block-trees for the purpose of approximate estimation in Section VII-B. If a graph is not weighted, we can assign all the weights to be one. In this case, finding a maximum weight B-width spanning block-tree is equivalent to finding a B-width spanning block-tree which retains the most number of edges in the final subgraph.

Algorithm 2 Constructing Maximum Weight Spanning Block-Trees

1: **procedure** MWSPANNINGBLOCKTREE(G, W, B)
2:     $\mathcal{G}$ = FindOptimalBlockTree(G) ; $\mathcal{G} = (\mathcal{C}, \mathcal{E})$
3:     $\mathcal{C}^B \leftarrow$ SplitCluster$(\mathcal{G}, W)$; See Section VII-A1
4:     $\mathcal{G}^B \leftarrow$ MWST$\left(\{C_{k_i}^{l_k}\}, W\right)$; See Section VII-A2
5: **end procedure**

Algorithm 2 outlines our approach to finding maximum weight spanning block-trees. The input to the algorithm is an undirected graph G, a weight matrix $W = [w_{i,j}]_n$, and the desired block-treewidth B. The output of the algorithm is a block-tree graph $\mathcal{G}^B = (\mathcal{C}^B, \mathcal{E}^B)$, where btw$(\mathcal{G}) \leq B$. Algorithm 2 is a greedy algorithm for finding spanning block-trees since solving this problem optimally is combinatorially complex. We first find the optimal block-tree graph $\mathcal{G}$ for the undirected graph G using the algorithm outlined in Section III-C (Line 2). The next steps in the algorithm are: (i) Splitting $\mathcal{C}$ into clusters $\mathcal{C}^B$ (Line 3), and (ii) finding edges $\mathcal{E}^B$ connecting two clusters so that $\mathcal{G}^B = (\mathcal{C}^B, \mathcal{E}^B)$ is a block-tree graph (Line 4).
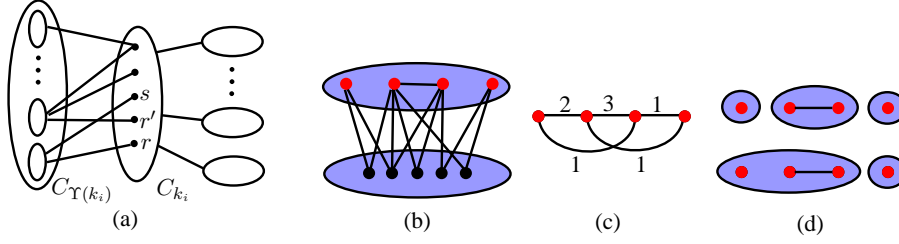
Fig. 5. (a) Notation used in Section VII-A1. (b) An example showing how clusters are split into smaller clusters. The goal is to split the cluster with red nodes into clusters with maximum size 2 and 3. All edges are assumed to have weight one. (c) Weighted graph constructed using η_{rs} given in (31). (d) Smaller clusters formed for $B = 2$ and $B = 3$. For $B = 2$, we first choose the middle two nodes since the weight between these two nodes in (c) is maximum. The remaining nodes are unconnected in (c), so we assign them to individual clusters. For $B = 3$, we again choose the middle two clusters first and then add another cluster so that the sum of weights is maximal.

1) Splitting Clusters: Since the maximum size of the cluster in $\mathcal{C}$ is greater than B, we first identify all clusters $\{C_{k_1}, \ldots, C_{k_m}\}$ such that $|C_{k_i}| > B$ for all $i = 1, \ldots, m$. Next, we split each C_{k_i} into smaller clusters so that $C_{k_i} = \{C^1_{k_i}, C^2_{k_i}, \ldots, C^h_{k_i}\}$, where $|C^j_{k_i}| \leq B$ for $j = 1, \ldots, h$. This splitting must be done in such a way the the nodes in the same cluster retain edges in the original graph with maximum weight. The algorithm we propose for splitting the clusters is as follows:

a) For each C_{k_i}, let $\Gamma(C_{k_i}) = \{C_{\Gamma_1(k_i)}, C_{\Gamma_2(k_i)}, \ldots, C_{\Gamma_q(k_i)}\}$ be all the clusters connected to C_{k_i} at the next scale and let $C_{\Upsilon(k_i)}$ be the cluster connected to C_{k_i} at the previous scale. We assume that we have already split $C_{\Upsilon(k_i)}$ such that $C_{\Upsilon(k_i)} = \{C^1_{\Upsilon(k_i)}, \ldots, C^w_{\Upsilon(k_i)}\}$, where $|C^j_{\Upsilon(k_i)}| \leq B$, for $j = 1, \ldots, w$. The notations introduced are shown in Fig. 5(a).

b) To split the cluster C_{k_i}, we first identify nodes in C_{k_i} which can be clustered together. For any two distinct nodes $r, s \in C_{k_i}$, if both r and s have edges in any one of the clusters $C^j_{\Upsilon(k_i)}$ for any $j = 1, \ldots, h$, then nodes r and s can be clustered together. For example, in Fig. 5(a), nodes r and r' can not be clustered together, whereas nodes r and s can be clustered together.

c) We now associate weights η_{rs} between nodes of C_{k_i} that can be clustered together:

$$\eta_{rs} = w_{rs} + \sum_{t \in \mathcal{N}(r) \cap \mathcal{N}(s) \cap \Gamma(C_{k_i})} (w_{rt} + w_{ts}) . \tag{31}$$

The intuition behind constructing the weights in (31) is to cluster nodes together that are connected to the same cluster at the next scale or are connected to each other. An example of constructing η_{rs} is shown in Fig 5(b) and Fig 5(c), where we assume the weights on each edge are one.

d) Using η_{rs} we construct a weighted graph on the nodes in C_{k_i}. To construct smaller clusters, we first
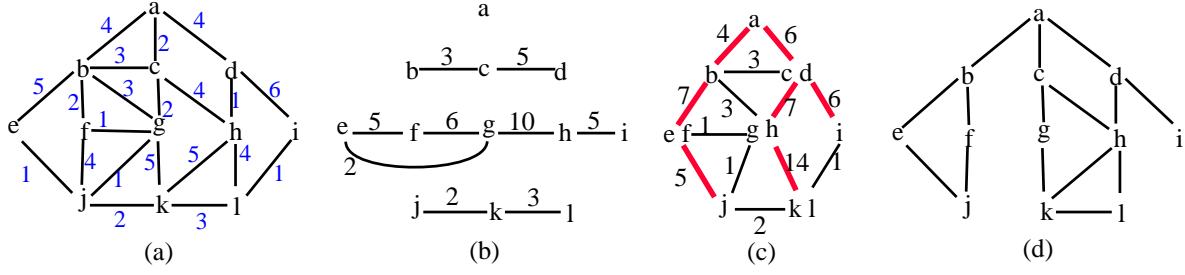
Fig. 6. Example of finding a maximum weight 2-width spanning block-tree using the weighted graph in (a). The clusters of the block-tree graph are $\{a\}, \{b, c, d\}, \{e, f, g, h, i\}, \{j, k, l\}$, shown in (b). Fig. 6(b) shows the weights η_{rs} for each cluster. Fig. 6(c) shows the smaller clusters obtained by retaining the edges with maximum weight in the weighted graph of (b). Note that the graph in (c) is a weighted block-tree graph. The red edges in (c) correspond to the maximum weight spanning tree. The final subgraph corresponding to the spanning block-tree in (c) is shown in (d).
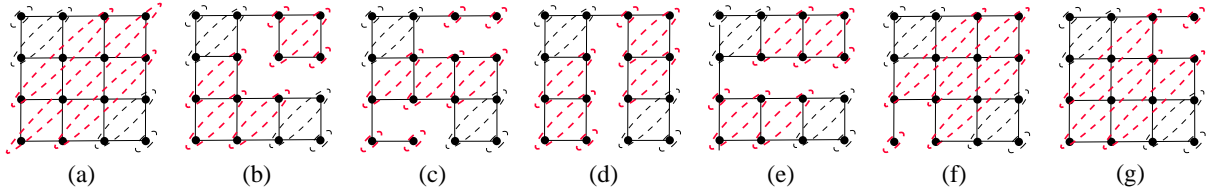

Fig. 7. (a) A 4×4 grid graph. (b)-(e) A collection of 2-width spanning block-trees. (f)-(g) A collection of 3-width spanning block-trees.

choose two nodes for which η_{rs} is maximum. We keep adding nodes to this cluster by choosing nodes connected to at least one node in this cluster until the cluster size is B. If no other node is connected to the new cluster, we start building another cluster. For example, given the weighted graph in Fig. 5(c), we construct clusters in Fig. 5(d) with $B = 2$ and $B = 3$.

After applying Steps (a)-(d) on all C_{k_i} such that $|C_{k_i}| > B$, we get the news clusters $\mathcal{C}^B$.

2) Find block-tree graph from clusters: Given the clusters $\mathcal{C}^B = \{C_k^B\}$, we can find a block-graph and associate weights between the clusters in $\mathcal{C}^B$ such that

$$w_{i,j}^B = \sum_{(i,j)\in(C_i^B \times C_j^B)\cap E} w_{i,j}. \tag{32}$$

' Equation (32) corresponds to the sum of all weights connecting two clusters C_i^B and C_j^B. Given the weights in (32), we can easily find a spanning block-tree using the maximum weight spanning tree (MWST) algorithms of Prim [46] or Kruskal [47]. Fig. 6 shows an example of using Algorithm 2 to find a maximum weight 2-width spanning block-tree. Fig. 7 shows a collection of spanning block-trees for a

4×4 grid graph.

3) Complexity: The complexity of Algorithm 2 depends on the structure of the graph. Assuming the optimal block-tree graph is given, the complexity of splitting the clusters is $O(|C_{k_i}|^2)$ for each C_{k_i} such that $|C_{k_i}| > B$. This number will be dominated by the cluster with maximum size, thus the complexity of splitting clusters is $O((\text{btw}(G))^2)$. The complexity of finding the final spanning block-tree depends on the graph and the number of edges in the block-graph formed using the clusters $\mathcal{C}^B$. In general, the complexity of this step decreases as B increases since this results in less number of edges and less number of smaller clusters $\mathcal{C}^B$. In practice, finding the optimal block-tree graph is hard, so we use the heuristic algorithm outlined in Section III-C.

B. Estimation Via Matrix Splitting

This Section reviews the matrix splitting approach to approximate estimation in Gaussian graphical models. For more details, see [8] and [45]. Let $\mathbf{x} \in \mathbb{R}^n$ be a Gaussian graphical model defined on a graph $G = (V, E)$ with covariance Σ and observations given by (28). The mmse estimate is given in (30). An alternate characterization of (30) is in the information form [8]:

$$V\widehat{\mathbf{x}} = H^T R^{-1} \mathbf{y} \tag{33}$$

$$V = (J + H^T R^{-1} H), \tag{34}$$

where $J = \Sigma^{-1}$ and $\widehat{P} = V^{-1}$ is the error covariance matrix. The matrices H and R are assumed to be diagonal, so the sparsity of V is the same as the sparsity of J. A family of approximate estimation algorithms, which are iterative algorithms, have been proposed in [8], with extensions in [45], [48]. The idea is to split the matrix V at each iteration k as $V = V_{S_k} - K_{S_k}$, where S_k is a subgraph of G. The sparsity of V_{S_k} corresponds to the sparsity of the subgraph S_k and the diagonals of V_{S_k} are the same as the diagonals V. Using matrix splitting, an iterative algorithm for estimation is given as [8]:

$$V_{S_k}\widehat{\mathbf{x}}^{(k)} = K_{S_k}\widehat{\mathbf{x}}^{(k-1)} + H^T R^{-1} \mathbf{y}, \tag{35}$$

where $\widehat{\mathbf{x}}^{(k)}$ is estimate at step k. If S_k is a subgraph with low treewidth or low block-treewidth, computing (35) is computationally tractable. Conditions for convergence of (35) are not known for general graphical models, however for walk-summable graphical models [49], convergence is guaranteed [45]. To compute the error covariance $\widehat{P}$, we can use the same matrix splitting approach to solve the linear system $V\widehat{P} = I_n$, where I_n is an $n \times n$ identity matrix [8].

It is clear that the choice of S_k in (35) plays an important role in the convergence of the algorithm. The problem of adaptively choosing S_k at each iteration was considered in [45], [50], where the authors proposed a weight matrix $w_{u,v}$, for $(u,v) \in E$, which signified the error reduction capacity of an edge (u,v) in the iterations (35):

$$w_{u,v}^{(k)} = \left(|h_u^{(k-1)}| + |h_v^{(k-1)}| \right) \frac{|J(u,v)|}{1 - |J(u,v)|} , \tag{36}$$

$$h_u^{(k-1)} = H^T R^{-1} \mathbf{y} - V \widehat{\mathbf{x}}^{(k-1)} . \tag{37}$$

Thus, at each iteration we want to choose a subgraph S_k such that the sum of all the weights in the graph is maximized while S_k is still a tractable subgraph. A popular approach is to use spanning trees, since finding an optimal spanning tree is efficient. However, as shown in [50], using tractable subgraphs, which are not trees, leads to faster convergence. Motivated by the need for algorithms with faster convergence, we propose to use spanning block-trees for approximate estimation of Gaussian graphical models. Thus, at each iteration we compute a weighted graph using (36) and then use these weights to compute a B-width spanning block-tree using Algorithm 2. In the next Section, we provide experimental results and show the improved convergence rates of estimation when using spanning block-tree graphs over spanning trees.

C. Experimental Results

Let $G = (V, E)$ be an undirected graph and suppose $\mathbf{x}$ is a Gaussian graphical model defined on G with covariance Σ and $J = \Sigma^{-1}$. We assume the diagonals of J are unity and let $S = I - J$. The non-zero entries in H correspond to the edges in the graph. To construct Gaussian graphical models, we choose the non-zero entries in S uniformly between $[-1, 1]$ and rescale S so that $\rho(\bar{S}) = 0.99$, where $\bar{S}$ is the matrix of absolute values of the elements of R and $\rho(\cdot)$ is the spectral radius. From [49], $\rho(\bar{S}) < 1$ ensures that the graphical model is walk-summable, which in turn ensures convergence of the iterative approximate estimation algorithm in (35) [45]. In all experiments, we assume that the observations are given by $y_s = x_s + n_s$, where $n_s \sim \mathcal{N}(0, 10)$. At each iteration, we compute the residual error, defined as $\frac{\|\mathbf{h}^{(n)}\|^2}{\|\mathbf{h}^{(0)}\|^2}$, where $\mathbf{h}^{(n)} = [h_1^{(n)}, \ldots, h_n^{(n)}]^T$, for $h_u^{(n)}$ defined in (37).

Fig. 8(a) shows results of doing estimation over a randomly generated 50×50 grid graph using spanning trees (Tree), spanning block-trees with block-width of three (BT-3), and spanning block-trees with block-width of five (BT-5). It is clear that using spanning block-trees leads to faster convergence. The same results hold for Fig. 8(b) that shows results on doing estimation over a randomly generated
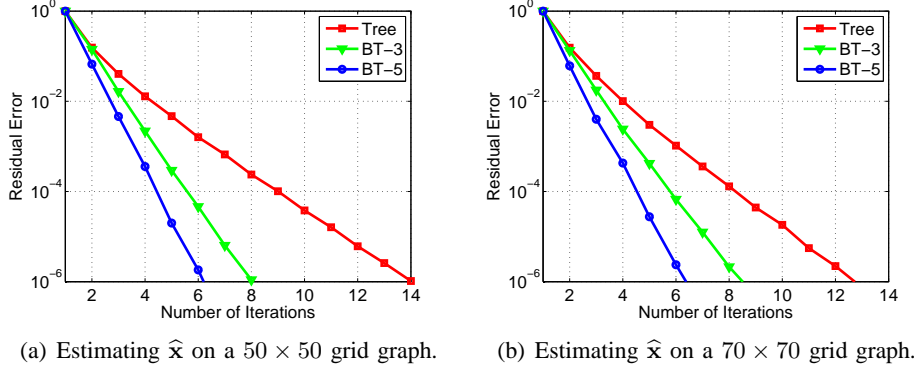
(a) Estimating $\widehat{\mathbf{x}}$ on a 50×50 grid graph.

(b) Estimating $\widehat{\mathbf{x}}$ on a 70×70 grid graph.

Fig. 8.   Normalized residual error when doing approximate estimation on a 50×50 grid graph and a 70×70 grid graph.

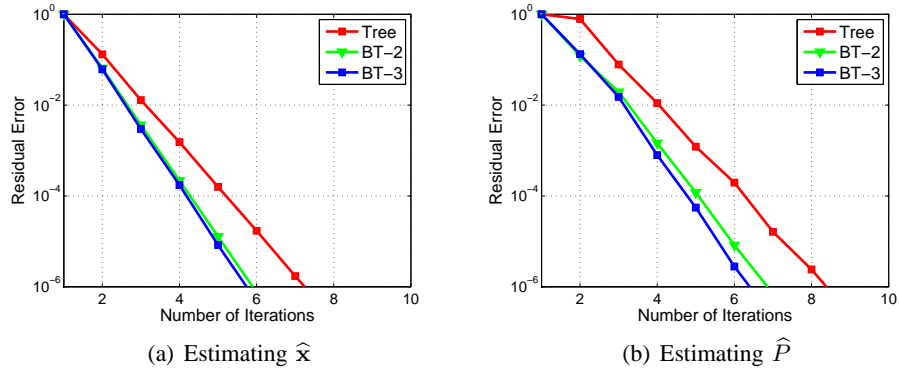(a) Estimating $\widehat{\mathbf{x}}$

(b) Estimating $\widehat{P}$

Fig. 9.   Normalized residual error when doing approximate estimation using spanning trees, spanning block-trees with block-width of two (BT-2), and spanning block-trees with block-width of three (BT-3) on a 15×15 grid graph with two nodes connected to all other nodes in the graph.

70×70 grid graph.

Fig. 9 shows results of doing estimation over a randomly generated 15×15 grid graph where two nodes are connected to all the nodes in the graph. Such graphs, where a few nodes have very high degree, are useful in video surveillance, modeling air traffic routes using hub-and-spoke model, or applications using small-world graphs. Fig. 9(a) plots the residual error at each iteration for the estimate and Fig. 9(b) plots the residual error at each iteration for the error covariance. Again, we observe that using spanning block-trees leads to faster convergence. The above simulations show that using spanning block-trees for approximate estimation is viable and leads to improved convergence speed.

VIII. SUMMARY

We introduced block-tree graphs as an alternative to junction-trees for constructing tree-structured graphs for arbitrary graphical models. We showed that constructing block-tree graphs is simple and

only requires information about a root cluster, which is a small number of nodes. On the other hand, constructing junction-trees requires knowledge of almost all nodes in the graph. For graphical models with boundary conditions, we showed that the block-tree graph framework leads to natural representations where we converted a boundary valued problem into a initial value problem. For Gaussian graphical models, the block-tree graph framework leads to state-space representations, using which we can easily recover recursive algorithms. Using the block-tree graph framework, we derived an algorithm for approximate estimation in Gaussian graphical models. The need for such algorithms arises because the problem of exact optimal estimation is computationally intractable for graphs with high treewidth. We proposed the use of spanning block-trees to derive approximate estimation algorithms for Gaussian graphical models. We showed that the speed of convergence when using spanning block-trees is faster when compared to using spanning trees. Further applications of spanning block-trees can be explored when doing approximate inference over discrete graphical models, using the results of [34], [51].

<div align="center">REFERENCES</div>

[1] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory and Applications (Monographs on Statistics and Applied Probability)*, 1st ed. Chapman & Hall/CRC, February 2005.
[2] M. J. Wainwright and M. I. Jordan, *Graphical Models, Exponential Families, and Variational Inference*. Hanover, MA, USA: Now Publishers Inc., 2008.
[3] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
[4] K. Chou, A. Willsky, and R. Nikoukhah, "Multiscale systems, Kalman filters, and Riccati equations," *IEEE Trans. Autom. Control*, vol. 39, no. 3, pp. 479 –492, mar 1994.
[5] K. Chou, A. Willsky, and A. Benveniste, "Multiscale recursive estimation, data fusion, and regularization," *IEEE Trans. Autom. Control*, vol. 39, no. 3, pp. 464 –478, mar 1994.
[6] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
[7] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*. Prentice Hall, 2000.
[8] E. B. Sudderth, M. J. Wainwright, and A. S. Willsky, "Embedded trees: estimation of Gaussian processes on graphs with cycles," *IEEE Trans. Signal Process.*, vol. 52, no. 11, pp. 3136–3150, Nov. 2004.
[9] N. Zhang and D. Poole, "A simple approach to Bayesian network computations," in *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, 1994, pp. 171–178.
[10] R. Dechter, "Bucket elimination: A unifying framework for reasoning," *Artificial Intelligence*, vol. 113, no. 1-2, pp. 41–85, Sep 1999.
[11] F. G. Cozman, "Generalizing variable elimination in Bayesian networks," in *Workshop on Probabilistic Reasoning in Artificial Intelligence*, 2000, pp. 27–32.
[12] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 50, no. 2, pp. 157–224, 1988.
[13] F. Jensen and F. Jensen, "Optimal junction trees," in *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*. San Francisco, CA: Morgan Kaufmann, 1994, pp. 360–36.
[14] N. Robertson and P. D. Seymour, "Graph minors. II. Algorithmic aspects of tree-width," *Journal of Algorithms*, vol. 7, no. 3, pp. 309 – 322, 1986.
[15] F. V. Jenson, S. L. Lauritzen, and K. G. Oleson, "Bayesian updating in causal probabilistic networks by local computation," *Computational Statistics Quarterly*, vol. 4, pp. 269–282, 1990.
[16] G. Shafer and P. P. Shenoy, "Probability propagation," *Annals of Mathematics and Artificial Intelligence*, no. 1-4, pp. 327–352, 1990.
[17] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM J. Algebraic Discrete Methods*, vol. 8, no. 2, pp. 277–284, 1987.

[18] H. L. Bodlaender and A. M. Koster, "Treewidth computations I. upper bounds," *Information and Computation*, vol. 208, no. 3, pp. 259 – 275, 2010.

[19] J. W. Woods and C. Radewan, "Kalman filtering in two dimensions," *IEEE Trans. Inf. Theory*, vol. 23, no. 4, pp. 473–482, Jul 1977.

[20] J. M. F. Moura and N. Balram, "Recursive structure of noncausal Gauss-Markov random fields," *IEEE Trans. Inf. Theory*, vol. IT-38, no. 2, pp. 334–354, March 1992.

[21] G. F. Cooper, "Nestor: A computer-based medical diagnostic aid that integrates causal and probabilistic knowledge," Ph.D. dissertation, Department of Computer Science, Stanford University, 1984.

[22] Y. Peng and J. A. Reggia, "Plausibility of diagnostic hypotheses," in *National Conference on Artificial Intelligence (AAAI'86)*, 1986, pp. 140–145.

[23] S. L. Lauritzen and N. Wermuth, "Graphical models for associations between variables, some of which are qualitative and some quantitative," *Annals of Statistics*, vol. 17, no. 1, pp. 31–57, 1989.

[24] M. Frydenberg, "The chain graph Markov property," *Scandinavian Journal of Statistics*, vol. 17, pp. 333–353, 1990.

[25] S. Andersson, D. Madigan, and M. Perlman, "An alternative Markov property for chain graphs," *Scandinavian Journal of Statistics*, vol. 28, pp. 33–85, 2001.

[26] J. E. Besag and P. A. P. Moran, "On the estimation and testing of spatial interaction in gaussian lattice processes," *Biometrika*, vol. 62, no. 3, pp. 555–562, 1975. [Online]. Available: http://www.jstor.org/stable/2335510

[27] R. Kashyap and R. Chellappa, "Estimation and choice of neighbors in spatial-interaction models of images," *IEEE Trans. Inf. Theory*, vol. 29, no. 1, pp. 60–72, Jan 1983.

[28] R. Chellappa and R. Kashyap, "Texture synthesis using 2-D noncausal autoregressive models," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 33, no. 1, pp. 194–203, Feb 1985.

[29] B. C. Levy, M. B. Adams, and A. S. Willsky, "Solution and linear estimation of 2-D nearest-neighbor models," *Proc. IEEE*, vol. 78, no. 4, pp. 627–641, Apr. 1990.

[30] D. Vats and J. M. F. Moura, "Telescoping recursive representations and estimation of Gauss-Markov random fields," *CoRR*, vol. abs/0907.5397, 2009.

[31] H. E. Rauch, F. Tung, and C. T. Stribel, "Maximum likelihood estimates of linear dynamical systems," *AIAA J.*, vol. 3, no. 8, pp. 1445–1450, August 1965.

[32] G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks (research note)," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 393–405, 1990.

[33] K. P. Murphy, Y. Weiss, and M. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Uncertainty in Artificial Intelligence*, 1999, pp. 467–475.

[34] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Map estimation via agreement on (hyper)trees: Message-passing and linear programming approaches," *IEEE Trans. Inf. Theory*, vol. 51, pp. 3697–3717, 2002.

[35] ——, "Tree-based reparameterization framework for analysis of sum-product and related algorithms," *IEEE Trans. Inf. Theory*, vol. 49, p. 2003, 2003.

[36] S. L. Lauritzen, *Graphical Models*. Oxford University Press, USA, 1996.

[37] J. Besag, "Spatial interaction and the statistical analysis of lattice systems," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 36, no. 2, pp. 192–236, 1974.

[38] J. M. F. Moura and S. Goswami, "Gauss-Markov random fields (GMrf) with continuous indices," *IEEE Trans. Inf. Theory*, vol. 43, no. 5, pp. 1560–1573, September 1997.

[39] M. J. Wainwright, "Stochastic processes on graphs: Geometric and variational approaches," Ph.D. dissertation, Department of EECS, Massachusetts Institute of Technology, 2002.

[40] L. L. Scharf, *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis*. Addison-Wesley, 1990.

[41] G. Verghese and T. Kailath, "A further note on backwards Markovian models (corresp.)," *IEEE Trans. Inf. Theory*, vol. 25, no. 1, pp. 121–124, Jan 1979.

[42] E. B. Sudderth, "Embedded trees: Estimation of Gaussian processes on graphs with cycles," Master's thesis, Massachusetts Institute of Technology, 2002.

[43] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Bethe free energy, Kikuchi approximations and belief propagation algorithms," 2000.

[44] D. Shahaf, A. Chechetka, and C. E. Guestrin, "Learning thin junction trees via graph cuts," in *Artificial Intelligence and Statistics (AISTATS)*, April 2009, pp. 113–120.

[45] V. Chandrasekaran, J. Johnson, and A. Willsky, "Estimation in gaussian graphical models using tractable subgraphs: A walk-sum analysis," *IEEE Trans. Signal Process.*, vol. 56, no. 5, pp. 1916 –1930, may 2008.

[46] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, p. 13891401, 1957.

[47] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, February 1956.

[48] V. Delouille, R. Neelamani, and R. Baraniuk, "Robust distributed estimation using the embedded subgraphs algorithm," *IEEE Trans. Signal Process.*, vol. 54, no. 8, pp. 2998 –3010, aug. 2006.

[49] D. M. Malioutov, J. K. Johnson, and A. S. Willsky, "Walk-sums and belief propagation in gaussian graphical models," *Journal of Machine Learning Research*, vol. 7, pp. 2031–2064, 2006.

[50] V. Chandrasekaran, J. Johnson, and A. Willsky, "Adaptive embedded subgraph algorithms using walk-sum analysis," in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, MA: MIT Press, 2008, pp. 249–256.

[51] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 10, pp. 1568–1583, 2006.